AF 2183

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re application of:	Eickemeyer et al.	:	Group Art Unit:	2183
Examiner:	Aimee J. Li	:	Serial No.:	09/894,260
Filed:	28 June 2001	:	Confirmation:	5055

Title: SHARED RESOURCE QUEUE FOR SIMULTANEOUS MULTITHREADING PROCESSING WHEREIN ENTRIES ALLOCATED TO DIFFERENT THREADS ARE CAPABLE OF BEING INTERSPERSED AMONG EACH OTHER AND A HEAD POINTER FOR ONE THREAD IS CAPABLE OF WRAPPING AROUND ITS OWN TAIL IN ORDER TO ACCESS A FREE ENTRY

Mail Stop Appeal Brief - Patents
Commissioner for Patents
P.O. Box 1405
Alexandria, VA 22313-1450

CERTIFICATE OF MAILING

I hereby certify that this correspondence is being deposited with the U.S. Post Office as first class mail with sufficient postage in an envelope addressed to Mail Stop Appeal Brief-Patents, Commissioner of Patents, P.O. Box 1450, Alexandria, VA 22313-1450 on 28 June 2005.

Karuna Ojanen, Reg. No. 32,484

Dear Sir:

APPEAL BRIEF IN SUPPORT OF APPEAL
FROM THE PRIMARY EXAMINER TO THE BOARD OF APPEALS

Applicant(s) herewith submit an appeal brief in support of the appeal to the Board of Appeals from the decision dated 12 January 2005 of the Examiner finally rejecting claims 1, 3-13, and 15.

The appeal brief fee of $500.00 is charged to Deposit Account No. **09-0465**. A duplicate copy of this sheet is enclosed.

An oral hearing is not requested.

07/05/2005 HGUTEMA1 00000016 090465 09894260
01 FC:1402 500.00 DA

Respectfully submitted,

By

Karuna Ojanen
Registration No. 32,484
507.282.0049 Voice
507.281,5722 Fax

28 June 2005
IBM Corporation
Intellectual Property Law
Dept. 917, Bldg. 006-1
3605 Highway 52 North
Rochester, MN 55901

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re application of:	Eickemeyer et al.	:	Date: 28 June 2005
Group Art Unit:	2183	:	IBM Corporation
Examiner:	Aimee J. Li	:	Intellectual Property Law
Serial No.:	09/894,260	:	Dept. 917, Bldg. 006-1
Filed:	28 June 2001	:	3605 Highway 52 North
Confirmation:	5055	:	Rochester, MN 55901

Title: SHARED RESOURCE QUEUE FOR SIMULTANEOUS MULTITHREADING PROCESSING WHEREIN ENTRIES ALLOCATED TO DIFFERENT THREADS ARE CAPABLE OF BEING INTERSPERSED AMONG EACH OTHER AND A HEAD POINTER FOR ONE THREAD IS CAPABLE OF WRAPPING AROUND ITS OWN TAIL IN ORDER TO ACCESS A FREE ENTRY

Mail Stop Appeal Brief - Patents
Commissioner for Patents
P.O. Box 1405
Alexandria, VA 22313-1450

Dear Sir:

CERTIFICATE OF MAILING

I hereby certify that this correspondence is being deposited with the U.S. Post Office as first class mail with sufficient postage in an envelope addressed to Mail Stop Appeal Brief-Patents, Commissioner of Patents, P.O. Box 1450, Alexandria, VA 22313-1450 on 28 June 2005.

Karuna Ojanen, Reg. No. 32,484

APPEAL BRIEF UNDER 37 CFR 1.192 TO THE BOARD OF PATENT APPEALS AND INTERFERENCES

REAL PARTY IN INTEREST

International Business Machines Corporation of Armonk, New York is the real party in interest and is the assignee of record of the entire interest, which assignment was recorded on 28 June 2001 at Reel 011970, frame 0894.

RELATED APPEALS AND INTERFERENCES

There are no related appeals and interferences.

STATUS OF CLAIMS

Claims 1, 4-13, and 15 are pending, are rejected, and are appealed.

Claims 2, 3, 14 were cancelled.

STATUS OF AMENDMENTS

An amendment dated 12 March 2005 was filed after a final rejection. Independent claim 1 was amended to incorporate the limitations of dependent claim 3; claim 3 was cancelled. Independent claim 10 and independent claim 11 were amended to include the limitation of claim 3 into the shared resource mechanism and the apparatus, respectively. Thus the amendments incorporated limitations of dependent claims into the independent claims. The Examiner's Advisory Action mailed 11 April 2005 was silent on whether the amendments would be entered for purposes of appeal.

Appellants believe that because the amendments after final incorporated the limitation of a dependent claim into independent claims, the amendments do not raise new issues that would require further consideration and/or search; nor do the amendments raise the issue of new matter. The amendments, moreover, place the application in better form for appeal by materially reducing or simplifying the issues for appeal. The Examiner's Advisory Action mailed 11 April 2005 as above, moreover, did NOT indicate that the amendments would not be entered.

SUMMARY OF CLAIMED SUBJECT MATTER

The invention of independent claim 1 and dependent claim 4, and independent claims 10 and the means plus function claims, independent claim 11 and dependent claim 12, is a shared resource queue within the hardware multithreaded processor core to support simultaneous multithreading. Independent claim 13 is the computer processing system having a hardware multithreaded processor core, as shown in Figure 2. Figure 2 also shows an out-of-order multithreaded processor and processing system of Figure 1 having a shared resource queue, the subject of dependent claim 5, independent claim 6, and dependent claim 15 is an out-of-order multithreaded processor and system. A multithreaded processor core has an instruction buffer 216 of Figure 2 as, *inter alia*, a means to fetch instructions from a plurality of threads and a means to distinguish instruction of one thread from another, as set forth on page 19, lines 15-16. Instructions are dispatched into a pipeline decode 218 as a means to decode the instructions and the rest of the pipeline 218-222 with instructions from more than one thread in various stages of the pipeline, as set forth in page 19, line 14 through page 20, line 22, including the means to dispatch the instructions, the dispatch unit 220 and issue queues 222, and the means to execute the instructions and resources, such as the fixed point unit 228, the floating point unit 230,t he load/store unit 232, the branch unit 226 and the condition register 224. The shared resource queue may be shared by multiple hardware threads and the shared resource queue provides input to the issue queues 222 of pipeline processor as shown in Figure 2 and discussed on page 20, lines 21 through page 21, line 14. Examples of shared resource queues include the branch information queue (BIQ) 252, load reorder queue (LRQ) 244, store reorder queue (SRQ) 246, and global completion table (GCT) 248 in a processor core as shown in Figure 2 and described on page 22 lines 10-14.

The shared resource queue itself is presented as reference numeral 300 in Figure 3 and each entry in the queue 310, 312, 314, 316, etc. has a thread id 340, a bank number 342, a valid marker 344, other content 346. For each thread in the shared resource queue, there is a head pointer 316 (for thread 2) and a tail pointer 322 (for thread 2) and perhaps a free entry 312 (for thread 2). The head pointer refers to the newest entry for that thread, page 23, line 8. The tail pointer indicates the oldest entry, page 24, line 5. The free pointer points to the next entry that the thread could use for allocation, page 23, line 6. As the head and tail pointers move in reaction to allocation and deallocation of entries, the head pointer can pass the tail pointer and the tail pointer can pass the head pointer, in either direction [means for a first entry of one thread to wrap around a last entry of the same thread], thus preventing unusable entries in the queue, page 24, lines 11-14. This may also be considered the means to indicate the number of times the first entry of the one thread wraps around the last entry of the same thread. In order for a thread to be able to use free entries after its tail pointer but before its head pointer, the inventors have created a bank number 342 per thread that counts the number of times the head pointer of a thread has passed its tail pointer.

The shared resource queue having a bank number that keeps track of how many times the head pointer passes its tail pointer offers many advantages. Out-of-order processing can occur without large amounts of precious processor core space being taken by duplicate registers for each thread and being unused. The flexibility offered by the shared resource queue permits all entries to be usable and yet enables the order of the instructions to be tracked. The versatility provided by the shared resources allows immediately-responsive dynamic processing so that one thread with a higher priority may continue processing or a thread that has stalled can be flushed quickly, without waiting for registers to be reloaded.

The invention of independent claim 7 and dependent claims 8 and 9 are method claims, and the steps of independent means-for claim 11 and dependent means-for claims 12 for the allocation, deallocation of, and flushing entries in the shared resource queue and are shown in Figures 5-11. Figure 6 and its description of the flow chart at page 25, line 26 through page 27, line 9 illustrates the method steps and the means of how to find a next entry pertaining to a particular thread in the shared resource queue. Figure 7 and its description at page 27, line 10 through page 28, line 10, describes the means for and the method by which to find the previous entry in the shared resource queue. Figure 8 and its description at page 28, line 11 through page 29, line 22 describe the method steps and the means to find a free entry in the shared resource queue. Figure 9 and its description at page 29, line 23 through page 31, line 3 describe the method steps and the means to allocate entries in the shared resource queue. Figure 10 and its description at page 31, line 4 through page 32, line 5 describe the method steps and the means to deallocate entries in the shared resource queue. Figure 11 and its description on page 32 through page 33, line 27 describe the method steps and the means to flush the resources pertaining to a particular thread in the shared resource queue.

GROUNDS OF REJECTION TO BE REVIEWED ON APPEAL

I. Whether claims 1, 3-4 and 7-9 are anticipated under 35 U.S.C. §102(e) by U.S. Patent No. 6,353,829 B1 entitled METHOD AND SYSTEM FOR MEMORY ALLOCATION IN A MULTIPROCESSING ENVIRONMENT to Koblenz et al. issued 05 March 2002 (herein referred to as Koblenz '829)?

II. Whether claims 5, 6, 10-13 and 15 are obvious under 35 U.S.C. §103(a) over the combination of Koblenz '829 and U.S. Patent No. 6,629,271 B1 entitled TECHNIQUE FOR SYNCHRONIZING FAULTS IN A PROCESSOR HAVING A REPLAY SYSTEM to Lee et al. issued 30 September 2003 (herein referred to as Lee '271) ?

ARGUMENT

A. Claims 1, 4 and 7-9 are not anticipated under 35 U.S.C. §102(e) by Koblenz '829.

A person shall be entitled to a patent unless - (e) the invention was described in ... (2) a patent granted on an application for patent by another filed in the United States before the invention by the applicant for patent .... 35 U.S.C. §102(e). It is well settled that a claim is anticipated if each and every limitation is found either expressly or inherently in a single prior art reference. In re Gurley, 27 F.3d 551, 31 USPQ.2d 1130, 1132 (Fed. Cir. 1994).

Appellants assert that the Examiner misread Koblenz '829 and that Koblenz '829 does not teach ***a resource queue*** having the limitation of a *head pointer and a tail pointer for at least one thread wherein the head pointer is the first entry of the at least one thread and the tail pointer is the last entry of the at least one thread, and* ***one of the unique resources is a bank number to indicate how many times the head pointer has wrapped around the tail pointer in order to maintain an order of the resources for the at least one thread,*** of independent claim 1. Koblenz '829 does not teach ***incrementing a bank***

number if the first entry passes a last entry of the particular thread before it finds the free entry, of independent claim 7.

The Examiner asserted that Koblenz '829 does teach the above limitations citing Koblenz '829 at column 11, lines 10-55; column 14, line 54 to column 15, line 25; column 16, line 37 to column 17, line 6; Figure 4; Figure 11; and Figure 14. *See* Examiner's Detailed Action mailed 12 January 2005 (herein referred to as the Final Rejection). In response, Appellants assert that Koblenz '829 does not teach a shared resource *queue* within the processor core; Koblenz' 829 teaches a memory system to allocate smaller blocks and larger blocks of memory, on the order of 10 megabytes, that are shared by multiprocessors, also capable of multithreading. Appellants carefully considered each and every figure, column, and line proffered by the Examiner in support of the alleged anticipation and ascertained that Koblenz '829 does not teach the claimed limitation of the "bank number" keeping track on when a head pointer passes a tail pointer in the same resource shared by multiple threads. *See* Appellants' response mailed 28 April 2005 (herein referred to as the Response to Final) on pages 13 through 15, reproduced here for convenience.

> Koblenz '829 teaches a data structure called a six list data structure that has a pointer to a circle portion having a number of items [warehouse header] currently in the circular list, and another pointer to a tail portion having a number of items [warehouse header] that have been removed from the circular list (column 11, lines 34-39). Each item [warehouse header] contains a next pointer and a last (or previous) pointer (column 13, lines 29-30), indicating the warehouse in memory to access data. It is unclear how many items are in the circular list - whether it is six, thirty-two, sixty-four; in any event, each item points to the next item, and in the embodiment shown, also has a pointer from the previous item.

Thus, this paragraph of Koblenz '829 only teaches a data structure having pointers to warehouses in memory.

> Column 11, lines 10-55 of Koblenz '829 discuss Figure 4 which is a block diagram of the bucket array and warehouse data structures. Each bin of the bucket array has a data structure having the variables of Table 1. The variables include a bin_tail pointer to the tail portion of the six list data structure and a

bin_circle pointer to the circle portion of the same six list data structure. A bin_highwater variable indicates the maximum number of lockers that have been simultaneously allocated for the warehouses in the bin; and the bin_highwater variable is incremented and decremented when one of the lockers is freed in the circle portion up to the highwater mark at which time a new warehouse is allocated. **The variables, however, do not include the number of times that bin_circle has wrapped around the bin_tail!** (Emphasis added)

Koblenz '829 at column 14, line 54 through column 15, line 25 discusses Figure 11 which is a flow chart of how to allocate a memory component. First, the routine returns a pointer to a block of memory and the determines if this block of memory is sufficient to satisfy the request. [Recall that we are talking about large and small blocks of memory, not must one entry in a queue.] If so, then the routine calculates the virtual bin number using a floating point technique and maps the virtual bin number to the actual bin number. The routine then retrieves a pointer and subtracts [adds -1] from the number of free lockers. If there are not a sufficient number of free lockers available in the bin, then a new warehouse of 64 lockers must be allocated. In any event, once a new warehouse is allocated, a warehouse header to a circle portion of the six list is added. If there are a sufficient number of free lockers, then the routine gets a locker from the existing warehouse. Again, the increment/decrement value refers only to the number of free lockers [bin_netfree] in a warehouse. **The circle pointer never passes the tail pointer and this is not recorded in a bank number, as applicants claim.** (Emphasis added)

Koblenz '892 at column 16, line 37 through column 17, line 6 discusses Figure 14 which is a flow chart of how an available warehouse header is retrieved from the tail portion of the six list. A description of the reallocation of next and previous pointers follows as the available warehouse header is retrieved from the tail portion and is linked into the circle portion. The routine adjusts the number of available lockers, and a new thread will take note that a warehouse header is available for use. **Again, Attorney for Applicants have not been able to discern that Koblenz ' 829 teaches that a bank number keeps track of the number of times a circle pointer passes the tail pointer.** (Emphasis added)

Respectfully, Table 1, Table 2, the description of the six list data structure with warehouse headers being allocated or freed simply do not teach or suggest the claimed "bank number to indicate how many times the head pointer/first entry has wrapped around the tail pointer/last entry in a resource queue" as Applicants claim in independent claims 1 and 7.

Because Appellants have rebutted the Examiner's assertion of anticipation, the Examiner now has the burden to state with particularity where the claim limitations are shown by the reference. The Examiner did not and, Appellants assert, cannot find the limitation of the "bank number" as claimed in the Koblenz '829 reference because it

does not exist in the teaching of the reference. Thus, the rejection must fall. Appellants request the Board to reverse the Examiner's rejection of claims 1, 4, 7-9 under 35 U.S.C. §102(e).

B. Claims 5, 6, 10-13 and 15 are not obvious under 35 U.S.C. §103(a) over the combination of Lee '271 and Koblenz '829.

The Examiner has then applied the method and apparatus of memory allocation of Koblenz '829 with an out-of-order processor architecture of Lee '271. It is well established that "[i]t is insufficient to establish obviousness that the separate elements of the invention existed in the prior art, absent some teaching or suggestion, in the prior art, to combine the elements." Arkie Lures, Inc. v. Gene Larew Tackle, Inc., 119 F.3d 953, 956-58, 43 USPQ.2d 1294 (Fed. Cir. 1997). The mere fact that teachings found in prior art could be combined as proposed by patent examiner [which in this case, the teachings cannot be combined to yield the claimed invention] does not make combination obvious absent some teaching, suggestion, or incentive supporting proposed combination; in present case, examiner's obviousness rejection must be reversed, since examiner failed to identify any such teaching, suggestion, or incentive to support proposed combination of two prior art references to achieve the claimed shared resource queue. *See* Ex parte Metcalf, 67 USPQ.2d 1633 (Bd. Pat. App. & Int. unpublished 02 May 2003). The suggestion to combine the reference must be in the prior art, not by the Examiner who, in this case, supplied the motivation for their combination. In the Final Rejection on page 7, the Examiner states that a "person of ordinary skill in the art would have recognized at the time the invention was made that out-of-order execution reduces idle cycles within a thread, thereby increasing the speed and efficiency of the device. Therefore, it would have been obvious ... to incorporate the out-of-order methods of Lee in the device of Koblenz to improve processor speed and efficiency." Appellants responded to the rejection by alleging that there is no motivation

in either reference for the combination, their combination would not yield the claimed invention, and, moreover, Koblenz' 829 teaches away from combination with the out-of-order processor architecture of Lee '271. Appellants will further discuss these points.

First, the Examiner misstates the law in the Advisory Action when she/he stated, "[n]ot teaching an element is not the same as teaching away from the element." In patent nomenclature, to 'teach away' means that 'a person of ordinary skill,' upon reading the reference, would be discouraged from following the path set out in the reference or would be led in a direction divergent from the path that was taken by the applicant. Monarch Knitting Machinery Corp. v. Sulzer Morat Gmbh, 139 F.3d 877, 855, (Fed. Cir 1998). A reference teaches away if it leave the impression that the product would not have the property sought by the applicant. In re Caldwell, 319 F.2d 254, 256, 138 USPQ 243, 245 (CCPA 1963). If there is no motivation to combine the references, the references teach away from each other. The prior art references relied upon must be considered in their entirety .... Disclosures in the references that diverge from and teach away from the invention cannot be disregarded. Dow Chemical Co. v. U.S. 20 Cl. Ct. 623, 630, 18 USPQ.2d 1654, 1662 (Cl. Ct. 1990). The presence or absence of motivation to combine references is a pure question of fact. Medical Instrumentation and Diagnostics Corp. v. Elekta AA, 68 USPQ.2d 1263 (Fed. Cir. 2003).

Appellants further assert that the Examiner misstated the facts in the Advisory Action when she/he stated "there is no explicit indication with Koblenz that out-of-order execution is undesirable and that only in-order execution should be used. The paragraph cited does not explicitly state anywhere that out-of-order execution is detrimental." But, indeed, Koblenz '829 does explicitly teach away from out-of-order processing; it specifically states the processor interleaves or switches threads on a cycle-by-cycle basis in order to maintain the order of instructions, i.e., "one thread of execution out of 128 streams supported by a processor executes on every clock cycle ... Thus, a new instruction for a different stream may be issued in each time period **without interfering with other instructions that are in the pipeline. When an**

instruction finishes, the stream to which it belongs becomes ready to execute the next instruction." Koblenz '829 at column 1, lines 28 through 41. (Emphasis added).

Perhaps, even more importantly, Koblenz '829 specifically teaches against shared resources for the threads. Koblenz '829 teaches separate resources for each thread: "each processor contains a complete set of registers for each stream" (Koblenz '829 at column 1, line 22-23); and the state of stream includes one 64-bit Stream Status Word ("SSW"), 32 64-bit General Registers ("R0-R31") and eight 32-bit Target Registers ("T0-T7"). Each MTA processor has 128 sets of SSWs, of general registers, of target registers. Thus, the state of each stream is immediately accessible by the processor **without the need to reload registers** when an instruction of a stream is to be executed." (Koblenz '829 at column 1, lines 41-48). Thus, there is an explicit indication of , i.e., a teaching away from, why Koblenz '829 would not be combined with Lee '271. One of skill in the art would not look to Lee '271 that teaches reloading a state of an executing thread that experiences a fault/exception from a duplicate pipeline when Koblenz '829 specifically states that each stream has its own sets of registers so that its state is immediately accessible by the processor without the need to reload?

Thus, Koblenz '829 teaches away from combining shared resources in the processor core of an out-of-order processor.

Lee '271, moreover, does not teach or proffer any suggestion for having shared resources within the processor core of an out-of-order processor. Lee '271 is concerned with faults which causes execution within a pipeline to stop. In order to recover the proper order of instructions in an out-of-order processor, Lee '271 teaches a duplicate pipeline having the original order of instructions so that when a fault occurs upon execution, the original order of the instructions can be recalled. The Examiner states that Lee '271 teaches a shared resource queue, i.e., teaches the claimed load reorder queue, the store reorder queue, the global completion table, the branch information queue, but herein lies another error of the Examiner. Lee '271 refers to a local cache miss at column 2, line 49; to a re-order buffer (ROB) having a copy of each instruction **in**

program order at column 4, line 66 through column 5, line 1; and to branch prediction in column 5, lines 33-41. None of these are shared resources having a head and tail pointer wherein a bank number counts how many times the head pointer passes the tail pointer.

So, if Lee '271 does not suggest shared resources because Lee '271 teaches a duplicate pipeline having the same instructions, and if Koblenz '829 does not teach or suggest out-of-order processing, and if neither suggest their combination, the Examiner has no basis in fact or in law to suggest their combination. Yes, Koblenz' 829 teaches away from a shared resource queue within an out-of-order processor, Koblenz '829 teaches that one instruction must finish before the stream becomes read to exedute the next instruction; on the other hand, an out-of-order processor allows an instruction from the same thread to complete before instructions ahead of it have completed (Applicants' specification at page 5, lines 14-15). And Lee '271 that preserves the instruction order to reload from the duplicate pipeline when an exception occurs teaches away from its combination with the shared memory allocation techniques of Koblenz '829.

CONCLUSION

Having thus proven that the Examiner erred in rejecting claims 1, 4, 7-9 under 35 U.S.C. §102(e) over Koblenz '829 because Koblenz' 829 does not teach a shared resource queue having a head pointer and a tail pointer and a bank number that counts how many times the head and tail pointer have passed one another, Appellants respectfully request the Board to reverse the Examiner and to allow claims 1, 4, 7-9.

Having further shown how the Examiner erred in applying the law of obviousness, and also how the Examiner erred in applying the facts under 35 U.S.C. §103(a): Koblenz' 829 teaches that in-order processing is important and only memory is shared, not the resource queues in the processor core; Lee '271 teaches recovery of instruction order with a duplicate pipeline. Even if combined, the combination does not

yield the claimed invention of a shared resource queue in which the bank number keeps track of how many times a head pointer passes a tail pointer. Appellants respectfully request the Board of Patent Appeals and Interference to reverse the Examiner's rejection of claims 5, 6, 10-13 and 15 under 35 U.S.C. §103(a) in view of the combination of Koblenz '829 and Lee '271.

Respectfully submitted,

RICHARD J. EICKEMEYER ET AL.

By _____

Karuna Ojanen
Attorney Reg. No. 32,484
507.282.0049 Voice
507.281.5722 Fax

IBM Corporation
Dept. 917, Bldg. 006-1
3605 Highway 52 North
Rochester, MN 55901

APPENDIX - REJECTED AND APPEALED CLAIMS 09/894,260

1. A resource queue, comprising:

(a) a plurality of entries, each entry having unique resources required for information processing;

(b) the plurality of entries allocated amongst a plurality of independent simultaneously executing hardware threads such that resources of more than one thread may be within the queue; and

(c) a portion of the plurality of entries being allocated to one thread and being capable of being interspersed among another portion of the plurality of entries allocated to another thread wherein a first entry of one thread is capable of wrapping around a last entry of the same thread to access an available entry;

(d) a head pointer and a tail pointer for at least one thread wherein the head pointer is the first entry of the at least one thread and the tail pointer is the last entry of the at least one thread, and

(e) one of the unique resources is a bank number to indicate how many times the head pointer has wrapped around the tail pointer in order to maintain an order of the resources for the at least one thread.

Claims 2-3 (Cancelled)

4. The resource queue of claim 1, further comprising:

(a) at least one free pointer for the at least one thread indicating an entry in the queue available for resources of the at least one thread.

5. The queue of claim 1, wherein the information processing further comprises:

(a) an out-of-order computer processor, and

(b) the resource queue may further comprise a load reorder queue and/or a store reorder queue and/or a global completion table and/or a branch information queue.

6. An out-of-order multithreaded computer processor, comprising:

(a) a load reorder queue;

(b) a store reorder queue;

(c) a global completion table;

(d) a branch information queue,

at least one of the queues being a resource queue comprising:

(i) a plurality of entries, each entry having unique resources required for information processing;

(ii) the plurality of entries allocated amongst a plurality of independent simultaneously executing hardware threads such that resources of more than one thread may be within the queue; and

(iii) a portion of the plurality of entries being allocated to one thread and being capable of being interspersed among another portion of the plurality of entries allocated to another thread;

(iv) a first entry of one thread being capable of wrapping around a last entry of the same thread;

(v) a head pointer and a tail pointer for at least one thread wherein the head pointer is the first entry of the at least one thread and the tail pointer is the last entry of the at least one thread;

(vi) a bank number to indicate how many times the head pointer has wrapped around the tail pointer in order to maintain an order of the resources for the at least one thread; and

(vii) at least one free pointer for the at least one thread indicating an entry in the queue available for resources of the at least one thread.

7. A method of allocating a shared resource queue for simultaneous multithreaded electronic data processing, comprising:

(a) determining if the shared resource queue is empty for a particular thread;

(b) finding a first entry of said particular thread;

(c) determining if the first entry and a free entry of the particular thread are the same;

(d) if, not advancing the first entry to the free entry;

(e) incrementing a bank number if the first entry passes a last entry of the particular thread before it finds the free entry;

(f) allocating the next free entry by storing resources for the particular thread.

8. The method of claim 7, further comprising deallocating multithreaded resources in the shared resource queue, comprising:

(a) locating the last entry in the shared resource queue pertaining to the particular thread;

(b) determining if the last entry is also the first entry for the particular thread;

(c) if not, finding the next entry pertaining to the particular thread;

(d) determining if the bank number of the next entry is the same as the last entry and if so, deallocating the next entry by marking the resources as invalid; and

(e) if not, then skipping over the next entry and decrementing the bank number;

(f) finding the next previous entry pertaining to the particular thread.

9. The method of claim 7, further comprising flushing the shared resource queue, comprising the steps of:

(a) setting a flush point indicative of an oldest entry to be deallocated pertaining to the particular thread; and

(b) invalidating all entries between a head pointer and the flush point which have the same and greater bank number than the bank number of the flush point.

10. A shared resource mechanism in a hardware multithreaded pipeline processor, said pipeline processor simultaneously processing a plurality of threads, said shared resource mechanism comprising:

(a) a dispatch stage of said pipeline processor;

(b) at least one shared resource queue connected to the dispatch stage;

(c) dispatch control logic connected to the dispatch stage and to the at least one shared resource queue; and

(d) an issue queue of said pipeline processor connected to said dispatch stage and to the at least one shared resource queue;

wherein the at least one shared resource queue allocates and deallocates resources for at least two of said plurality of threads passing into said issue queue in response to the dispatch control logic and the at least one shared resource queue further comprises a plurality of entries allocated to one thread and capable of being

interspersed among another plurality of entries allocated to another of the plurality of threads wherein a bank number records the number of times a first entry of one thread wraps around a last entry of the same thread to access an available entry for allocating resources of the one thread.

11. An apparatus to enhance processor efficiency, comprising:

(a) means to fetch instructions from a plurality of threads into a hardware multithreaded pipeline processor;

(b) means to distinguish said instructions into one of a plurality of threads;

(c) means to decode said instructions;

(d) means to allocate a plurality of entries in at least one shared resource between at least two of the plurality of threads simultaneously executing;

(e) means to allocate and intersperse entries in the at least one shared resource to one thread among entries allocated to other threads;

(f) means for a first entry of one thread to wrap around a last entry of the same thread;

(g) means to indicate the number of times the first entry of the one thread wraps around the last entry of the same thread;

(h) resources and at least one shared resource queue for dispatching said instructions;

(i) means to dispatch said instructions;

(j) means to deallocate said entries in said at least one shared resource when one of said at least two threads are dispatched;

(k) means to execute said instructions and said resources for the one of said at least two threads.

12. The apparatus of claim 11, further comprising:

(a) means to flush the at least one shared resource of all of said entries pertaining to the one of said at least two threads.

13. A computer processing system, comprising:

(a) a central processing unit;

(b) a semiconductor memory unit attached to said central processing unit;

(c) at least one memory drive capable of having removable memory;

(d) a keyboard/pointing device controller attached to said central processing unit for attachment to a keyboard and/or a pointing device for a user to interact with said computer processing system;

(e) a plurality of adapters connected to said central processing unit to connect to at least one input/output device for purposes of communicating with other computers, networks, peripheral devices, and display devices;

(f) a hardware multithreading pipelined processor within said central processing unit to simultaneously process at least two independent

threads of execution, said pipelined processor comprising a fetch stage, a decode stage, and a dispatch stage; and

(g) at least one shared resource queue within said central processing unit, said shared resource queue having a plurality of entries pertaining to more than one thread in which entries pertaining to different threads are interspersed among each other, and a head pointer pertaining to an entry of one thread is capable of wrapping around a tail pointer pertaining to another entry of the same one thread to access an available entry and the number of times the head pointer wraps around the tail pointer is recorded.

14. (Cancelled)

15. The computer processor of claim 13, wherein the hardware multithreaded pipelined processor in the central processing unit is an out-of-order processor.